Intelligent Systems Project: Question Answering Machine

Matheus Galvão - a54185,

Luan Klein - a54160,

Matteo Prina - m312574

Supervisors:

Profº. Dr. Rui Lopes

Bragança

2022-2023

Contents

List of Figures

Chapter 1

Introduction

Humans have always dreamt of holding conversations with machines since the invention of the computer. Examples of this are the Turing Test and the innumerous sci-fi stories developed over the years where computers are portrayed as conscious beings who can talk, make decisions and answer any question. The future of intelligent machines promised by Hollywood is not a reality (yet), but the academy and industry have made significant progress in the field of NLP (Natural Language Processing).

Systems capable of understanding human language were developed and integrated into the daily lives of billions of people, virtual assistants like Apple's Siri, Amazon's Alexa and Microsoft's Cortana are some of the most popular examples of NLP applications used today. Question Answering (QA) systems specifically are applications designed to answer questions made in natural human language, such a system is a popular Google search feature, integrated to show simple answers, many times correct, to questions made on its search engine by reading numerous web pages and finding a possible answer.

This work has the objective of developing a Question Answering Machine based on the second version of the Stanford Question Answering Dataset (SQuAD 2.0), which consists of questions posed on a set of Wikipedia articles, where each question can be either unanswerable or a segment of text from the corresponding reading passage. In order to achieve the objective, the project aims to explore the best approaches currently used for the problem, understand the architecture used by the best models, and find ways

to build a QA machine considering the limited processing capabilities available, which was accomplished by integrating a pre-fine-tuned transformer model from the Hugging Face repository [1], which achieves highly accurate results.

The paper is composed of: Section 2 presents the description of the project, with some theoretical background of the models used, an introduction to the data, the training process, the evaluation method, and the methodology applied for the system development; Section 3 presents the results and discussions; and finally, Section 4 presents the conclusions.

Chapter 2

Project Description

This chapter accounts for the methodology used for the system development, explaining what kind of system it is based on, the architecture chosen, how to overcome our limited processing capabilities, the data used, and the algorithm developed.

2.1 Information Retrieval-Based Systems

Information retrieval-based (IR) QA systems are algorithms designed to harness information from unstructured data, such as text. IR systems find and extract a text segment from a large collection of documents, by identifying the most relevant texts and extracting an answer from their contents using a document reader algorithm [2]. A system based on information retrieval is a good fit for our problem, as it needs to find answers from Wikipedia passages.

2.2 Transformers

Neural Network models can produce exceptional results when applied as a document reader, the Transformer architecture is widely used for such applications. Transformer models aim to solve sequence-to-sequence tasks by applying self-attention techniques, which allow the neural network to understand a word in the context of words around it,

this kind of model is also based on the encoder-decoder architecture, as seen in Figure 2.1, where the encoder steps through the input time steps and encodes the entire sequence into a fixed-length vector, while the decoder steps through the output time steps while reading from the vector [3]. Among the advantages held by this architecture are the potential to understand the relationship between sequential elements far from each other, equal attention to all the elements in the sequence, and the ability to process and train more data in less time than other NLP models [2]. Taking into consideration the many advantages of this architecture, its popularity in NLP problems, and the vast number of pre-trained models publicly available for several different datasets, this work uses a Transformer model for the development of a QA machine.

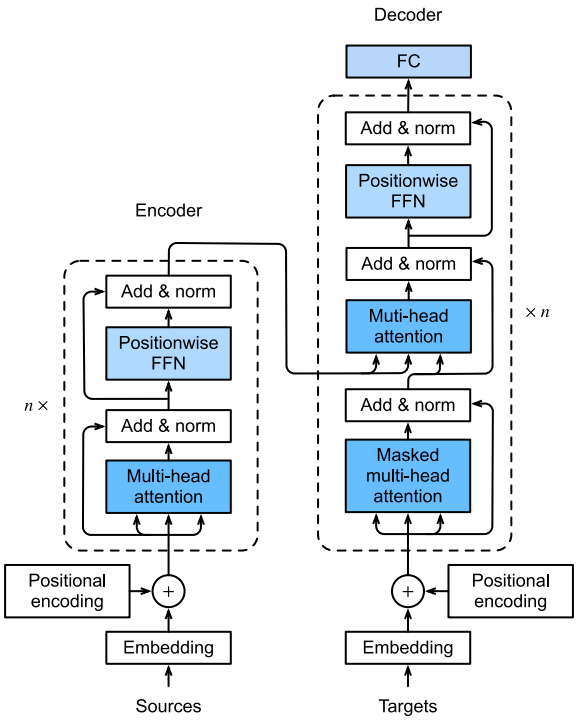

Figure 2.1: Transformer architecture. Source [4].

2.3 Pre-trained and Fine-tuned Models

For freshly created models to learn natural language is difficult and takes huge amounts of time, for that reason many applications choose to use pre-trained models. It is possible to access numerous pre-trained NLP models in public repositories, such as Hugging Face [1], where diverse models have been fed by massive unsupervised training sets in order to learn basic language modeling. After choosing a NLP model, it can be fine-tuned to work with specific datasets, for instance, a complex fine-tuning script is provided by [1] to tune Transformer models on one of the SQuAD datasets, though it is not advisable to use in the absence of advanced GPUs and a considerable amount of RAM, as each epoch can take several hours to process even in powerful machines [2].

One alternative for the development of a QA machine is to use models already fine-tuned, doing so it is not necessary to manually fine-tune the model, which would be an extremely hard task with our available computational resources. Several different advanced transformer models fine-tuned for the SQuaD dataset are also publicly available in [1], which made the development of an accurate QA machine possible for this project. Next, several approaches to NLP will be brief discussing, where three of these approaches were used in the present work.

2.3.1 GPT-3

Generative Pre-trained Transformer 3, as known as GPT-3 is a model developed by OpenAI [5], it is one of the state-of-the-art models on NLP. This model has been fed around 570GB of text information gathered by crawling the internet. This model contains around 175 billion hyperparameters, which demonstrates how big and complex the model is. GPT-3 has gained notorious evidence in the world and an interesting article was published by Forbes talking about this model [6]. Once the GPT-3 will not be used in this work, deeper concepts of this model will not be detailed either, leaving the reader free to consult the references cited above.

The text bellow is the answer of the OpenAI platform[1] (using GPT-3) for the question *What is GPT?*:

> GPT (Generative Pre-training Transformer) is a large language model developed by OpenAI that has been trained on a vast amount of data in order to generate human-like text. GPT was designed to be able to generate text in a variety of styles and formats, including news articles, stories, poems, and more.
>
> One of the key innovations of GPT is that it is a "transformer" model, meaning that it is able to process input sequences efficiently and in parallel, making it well-suited for large-scale tasks. GPT is also "generative," meaning that it is able to generate text by predicting the next word in a sequence, given the words that come before it.
>
> Since its release in 2018, GPT has become one of the most widely-used and influential language models, and it has been used to generate text for a variety of applications, including language translation, chatbots and automated content generation.

2.3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) was presented in 2018 by researchers at Google AI Language [7]. In the BERT model, the training occurs in two different chunks: The first is the pre-training and the second is the fine-tuning. It is possible to simplify the explanation of these two parts by saying that the first main goal is to teach the model to understand the natural language and the second is to specify the model in a task, for example, question answering. The core of the BERT model is composed of a stack of encoders, which aims to learn the language. The idea the fine-tuning is to just add an output layer and train the parameters of this layer.

The input of the BERT model is composed of three main parts: positional encoding (which represents the position of the word in a sentence); sentence encoding (used when there are multiple sentences at the same time to define if a word belongs to the first or to the second sentence, for instance, in question answering); and the token encoding (that represents the token of every single word in a numerical way). Furthermore, BERT has two special tokens: [CLS] and [SEP]. It is important because BERT can take as input

[1] https://openai.com/blog/chatgpt/

either one or two sentences, and uses the special token [SEP] to differentiate them while [CLS] token always appears at the start of the text [8].

To train the model to understand natural language, BERT does two different tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In a simple way, some words are hidden in an MLM task, and the model has to discover the correct word. For example, in the sentence "*A robot should not hurt humans*", the word *not* can be replaced for a special marker, and the new sentence is "*A robot should [MASK] hurt humans*", and the model has to discover the word replaced. Around 15% of the words are hidden in this train. On the other hand, in the NSP training two sentences are given to the model, and it has to decide if the second sentence is logically connected with the first one. For example, suppose the first sentence is "*How are you feeling today?*", if the second sentence is "*I'm ok, thank you!*", the model has to recognize this sentence has a connection with the first one, but if the second sentence is "*And blue was red!*" does not have a logical connection [9].

Furthermore, it is important to highlight the different types of BERT. The BERT base is composed of 12 layers with a hidden size of 768 and 110 million parameters, while the BERT large is composed of 24 layers and 340 million parameters[2].

The BERT model was trained using only unlabeled data, using text from the entire English Wikipedia and the Brown Corpus. Moreover, the model continued learning through unsupervised methods using data from other applications, such as Google searches [10]. The model used in this work is present on the Hugging Face platform and is already fined-tuned for the second version of the SQuAD dataset[3].

2.3.3 RoBERTa

RoBERTa, an acronym to Robustly Optimized BERT Pretraining Approach, is a model based on BERT and presented by [11]. Built on top of BERT and with modified key hyper-parameters, it removes the next-sentence pre-training objective, and trains with

[2]More at: https://huggingface.co/transformers/v2.2.0/pretrained_models.html
[3]https://huggingface.co/deepset/bert-base-cased-squad2

much larger mini-batches and learning rates [12]. This model is composed of 24 layers, with hidden sizes of 1024 and 355 million parameters. The model used in this work is present on the Hugging Face platform and is already fined-tuned for the second version of the SQuAD dataset[4].

From the Hugging Face platform [12], some important aspects of RoBERTa are:

- RoBERTa implementation is the same as the BERT model with small embedding tweaks, along side a setup for Roberta pre-trained models.
- RoBERTa has the same architecture as BERT, but uses a byte-level BPE as a tokenizer (same as GPT-2) and uses a different pre-training scheme.
- RoBERTa does not have token_type_ids, so you don't need to indicate which token belongs to which segment. Just separate your segments with the separation token tokenizer.sep_token (or </s>)

2.3.4 DeBerta

DeBERTa, an acronym of Decoding-enhanced BERT, is a model presented in 2020 by [13]. The DeBERTa V3 large is composed of 24 layers, a hidden size of 1024, 304 million parameters, and it was trained using 160GB of data. This model substitutes the MLM process training with replaced token detection (RTD), a more sample-efficient pre-training task. The model used in this work is present on the Hugging Face platform and has already been fine-tuned for the second version of the SQuAD dataset, and this pre-finel model was used in this work[5].

2.4 Data

The data used for this project was the Stanford Question Answering Dataset (SQuAD), which is a reading comprehension dataset, consisting of questions posed by crowdworkers

[4]https://huggingface.co/deepset/roberta-base-squad2
[5]https://huggingface.co/deepset/deberta-v3-large-squad2

on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable [14].

The second version of Squad (SQuAD2.0) combines the 100,000 questions in SQuAD1.1 with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on SQuAD2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering [14].

The data used is composed of two json files, train-v2.0 and dev-v2.0, for training and evaluation respectively. In a briefly way, the dataset is composed by a large number of questions, which are divided in topics. Inside each topic, a few questions are related with the same context (the context is a text where the model will search for the answer for the question). Furthermore, each questions has some answers (i.e., the possible answers to that question) and if the question is impossible to answer using the context, this field is empty. Moreover, an important element in the dataset on each question are the *answer_start* and the *answer_end*, which are the index where the question answer starts and ends.

2.5 Train

To train the BERT by fine-tuning it to the Squad dataset, several tutorials are available on the internet. The idea of this training is to specialize the BERT model to answer questions, which is called adding a *head* to the model. One of the followed tutorials was [15]. It is possible briefly describe the training process as: Getting the data (questions, context, and answers), preparing the data (for example, each answer has a start and an end), tokenize it, getting the token in the correct form (in the present paper, PyTorch was used), setting up the parameters of the training (such as learning rate), and in a loop, feed the model with the data, calculate the error and do the training. More details about this are available on the reference.

In addition, it is important to highlight when such training was attempted, it presented to be a process too long to complete, taking more than 100 hours per epoch. Consequently, other approaches were taken, which the training was no necessary anymore, by using pre-trained models, and no more focus was given to training.

2.6 Evaluation Method

To evaluate the quality of the models two main metrics were used: Exact match (EM) and the F-1 score. The EM, as the name suggests, measures the proportion of questions where the predicted answer is identical to the correct answer. On the other hand, the F1-score is more flexible than the EM score (it is possible say the F1-score is more closely resembles human judgment as far as the similarity of two answer strings) [16].

To generate these metrics, the code used was from the Official evaluation script for SQuAD version 2.0, and the data used to answer the question was the Dev Set v2.0, both are the official from The Stanford Question Answering Dataset and are available [14].

2.7 Methodology

To build the Answering Question Machine, the language used was Python. The system was previously developed in a Jupyter Notebook, but for the final submission, a .py (file extension for python) was used. The first task done in this work was to find the official data (training dataset, test dataset and the official evaluation script), all these files are available on the official platform, as presented above.

The next task was to develop the system, which loaded a fine-tuned model and fed it with the questions (along the respective context) to generate the answers. To do this, the transformers library was used (version 4.24.0), two elements in specific: *AutoTokenizer* and *AutoModelForQuestionAnswering*. The first one is used to convert the strings to tokens (according to the selected model) and the second loads the model and receives the tokens, answering the question. Furthermore, the Pytorch (version 1.12.1) framework was

used to get the answers. More details about the code are available in the comments of the code.

It was also possible to fine-tuned a pre-trained BERT model. Some tutorials were used to understand how this process works. A file train.py was developed to train the model using the official SQuAD 2.0 dataset. Although, a problem was found: The training time. A huge amount of computational power and many GBs of RAM was necessary, aspects that the students' computers do not possess. Using, for example, a computer with a processor *AMD EPYC 7351 16-Core Processor (2.40GHz)* and 32 GB of RAM memory (without GPU), each epoch of the train would spend around 165 hours, that is a lot of time, even for few epochs.

Once this scenario was present, several pre-fined models on the SQuAD 2.0 are available on the Hugging Face platform. So, instead of training a model by ourselves (which would consume a lot of computational power and time, and probably the results were not as good as the other pre-fined models trained with powerful computational resources) three different models were selected, tested, and the model that presents the best result was considered for the final submission. The models are: *bert-base-cased-squad2*, *roberta-base-squad2* and *deberta-v3-large-squad2*. To evaluate each model, the official evaluation model was used, as explained above.

Finally, after these steps, a Dockerfile was composed. The Docker image built was based on Python 3.9, the necessary libraries were installed through PIP (all the libraries are present in the requirements.txt), and finally executing the file main.py, which loads the model and answers the questions. An important aspect is that even with the pre-trained model, this process takes a long time to execute because of the numerous questions to answer on the validation set (around 11800 questions). The approximate time observed was around 10 hours. For that reason, the final submission has two different Dockerfiles, one that runs the code to answer all the questions and calls the evaluation script (the name of the docker file is *dockefile-generate.dockerfile*), and another Dockerfile (with the dafault name, *Dockerfile*) that only executes the evaluation script with the answers already provided by the group, doing so, it is not necessary to wait for the whole execution of the

main file. It is important that to correctly execute the code that generates the predictions, the Docker container requires at least 8 GB of RAM, otherwise, the container will crash during will crash. All the instructions on how to build the images and run the containers are specified in the README.md file.

Chapter 3

Results and Discussions

Three different transformer models were applied and evaluated during the development of this project, *bert-base-cased-squad2*, *roberta-base-squad2*, and *deberta-v3-large-squad2*, all of them based on the BERT model. As shown in the Table 3.1, the best performance was obtained from the *deberta-v3-large-squad2* model, obtaining a F1 score of 0.9087, a value significantly superior compared to the other two models, such score is considered high and close to the best score ever obtained to the SQuAD 2.0 dataset, which is 0.93214 [14]. Although the deberta model presented the best results in our tests, the processing time required also is far higher than from the other models, almost 6 times more, which can be a disadvantage if trying to run on weak machines. The roberta model is capable to produce great results, with an F1 score of 0.825 in much less time, being a fast and efficient model.

Model	**Exact match (EM)**	**F1**	**Average Time (per question)**
bert-base-cased-squad2	66.91%	71.33%	0.42
roberta-base-squad2	79.48%	82.50%	0.41
deberta-v3-large-squad2	87.80%	90.87%	2.30

Table 3.1: Models results

Chapter 4

Conclusion

Building a QA machine is a hard problem to face, however many technologies, architectures, and models have evolved to make such an objective possible, it has proven to be an extremely expensive task computationally to solve using the best tools available, which were found to be transformers models. Although a difficulty like the one mentioned exists, many other researchers have worked on it and developed models fine-tuned to the dataset of our problem with much more capable machines. Therefore, the use of fine-tuned models accessible in public repositories has proven to be the best option to build an accurate system with the computational resources available.

Finally, the objective of developing a QA machine was accomplished through the exploration and application of pre-trained models, achieving performances very close to the state-of-the-art of the SQuAD dataset. Our evaluation considered the models' performances, but also their time to produce an answer, since, in the case of building a QA application for the general public, it could be a factor of relevance. Future progress in this work could attain better results with more computational resources, which would make possible and much faster other attempts with different models and with the execution of our own training.

Bibliography

[1] *Hugging face – the ai community building the future.* https://huggingface.co/, (Accessed on 12/21/2022).

[2] *Building a qa system with bert on wikipedia | nlp for question answering*, https://qa.fastforwardlabs.com/pytorch/hugging%20face/wikipedia/bert/transformers/2020/05/19/Getting_Started_with_QA.html, (Accessed on 12/24/2022), 2020.

[3] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, 2022.

[4] *11.7. the transformer architecture — dive into deep learning 1.0.0-beta0 documentation*, https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html, (Accessed on 01/06/2023).

[5] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[6] B. Marr, *What is gpt-3 and why is it revolutionizing artificial intelligence?* Dec. 2021. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2020/10/05/what-is-gpt-3-and-why-is-it-revolutionizing-artificial-intelligence/?sh=40790359481a.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] D. Dhami, *Understanding bert-word embeddings*, Jul. 2020. [Online]. Available: https://medium.com/@dhartidhami/understanding-bert-word-embeddings-7dc4d2ea54ca.

[9] AssemblyAI, *What is bert and how does it work? | a quick review*, Jan. 2022. [Online]. Available: https://www.youtube.com/watch?v=6ahxPTLZxU8&list=RDCMUCtatfZMf-8EkIwASXM4ts0A&start_radio=1&t=527s&ab_channel=AssemblyAI.

[10] B. Lutkevich, *What is bert (language model) and how does it work?* Jan. 2020. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model.

[11] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[12] H. Face, *Roberta*. [Online]. Available: https://huggingface.co/docs/transformers/model_doc/roberta.

[13] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[14] Stanford, *Squad2.0*. [Online]. Available: https://rajpurkar.github.io/SQuAD-explorer/.

[15] J. Briggs, *How to train bert for qamp;a in any language*, Oct. 2021. [Online]. Available: https://towardsdatascience.com/how-to-train-bert-for-q-a-in-any-language-63b62c780014.

[16] A. A., *Metrics to evaluate a question answering system*. [Online]. Available: https://www.deepset.ai/blog/metrics-to-evaluate-a-question-answering-system.